Managing network congestion with a Kohonen-based RED queue

Emmanuel Lochin
Université de Toulouse, DMIA
ISAE - France,
emmanuel.lochin@isae.fr

Bruno Talavera
Université Pierre et Marie Curie,
Polytech'Paris, France,
bruno.talavera@upmc.fr

***Abstract*— The behaviour of the TCP AIMD algorithm is known to cause queue length oscillations when congestion occurs at a router output link. Indeed, due to these queueing variations, end-to-end applications experience large delay jitter. Many studies have proposed efficient Active Queue Management (AQM) mechanisms in order to reduce queue oscillations and stabilize the queue length. These AQM are mostly improvements of the Random Early Detection (RED) model. Unfortunately, these enhancements do not react in a similar manner for various network conditions and are strongly sensitive to their initial setting parameters. Although this paper proposes a solution to overcome the difficulties of setting these parameters by using a Kohonen neural network model, another goal of this study is to investigate whether cognitive intelligence could be placed in the core network to solve such stability problem. In our context, we use results from the neural network area to demonstrate that our proposal, named Kohonen-RED (KRED), enables a stable queue length without complex parameters setting and passive measurements.**

I. Introduction

More than ten years ago, the Random Early Detection (RED) was proposed to avoid congested links [5]. The main idea of the RED algorithm is to drop packets before the queue is full. As a consequence, when a TCP source gets such preventive drops, it decreases the emitted throughput according to the AIMD (Additive Increase Multiplicative Decrease) algorithm. RED drops packets with an increasing probability (max_p) when the occupancy of the queue lies between two thresholds (min_{th}, max_{th}). The goal of RED is to maintain a small buffer occupancy and avoid casual bursts of packet losses.

The authors in [10] and [11] weight up the disadvantages for deploying such mechanism. In certain cases, increasing the number of dropped packets can have unexpected effects on the overall performance [11]. This has motivated the use of preventive marking instead of preventive dropping with the use of the Efficient Congestion Notification (ECN) flag. In this case, instead of dropping packets, the RED queue marks the packet's ECN flag to notify senders that they are crossing a congested link and that they should decrease their sending rate. In [10], the authors claim that tuning parameters in RED remains an inexact science. We fully acknowledge the criticisms of this approach which motivate our proposal of managing the RED configuration with a neural network.

Feng RED (FRED) [2] and Adaptive RED (ARED) [4] introduced the notion of adaptive AQM. These adaptive strategies recompute the max_p probability value following an AIMD algorithm. However, the parameters that weight this AIMD process remain difficult to estimate.

Some past work have already suggested that RED is fundamentally hard to tune [8]. In this study, the authors show that RED parameters can be tuned to improve stability, but only at the cost of large queues even when they are dynamically adjusted. Even if other different queueing approaches have been proposed to improve the efficiency of RED-like algorithms in various network conditions, the parameters used to set these new AQM are sometimes more complex to determine than RED. In particular, this is the case for the PI controller [6]. Nowadays, general parameters able to stabilize the queue don't yet exist whatever the AQM used and we could discuss whether the problem is in fact solvable.

Although the validity of RED concept is still debated, we claim that the parameters' settings are one of the main barrier to its acceptance. In this paper, we propose to compute the optimal max_p value with a Kohonen neural network [7]. We do not attempt to design another queueing mechanism or propose to enhance the core mechanism itself. We only focus on the optimal estimation of the probability parameter. This paper aims at illustrating the impact of the role of learning mechanisms on core network Internet problems with similar motivation than the one presented in [1].

This paper is structured as follow. Section II presents the motivation of this work. Section III gives pointers related to the implementation of the core mechanism. Section IV presents the training phase of the neural network. Then, section V evaluates the proposal and finally section VI gives the perspectives of this work.

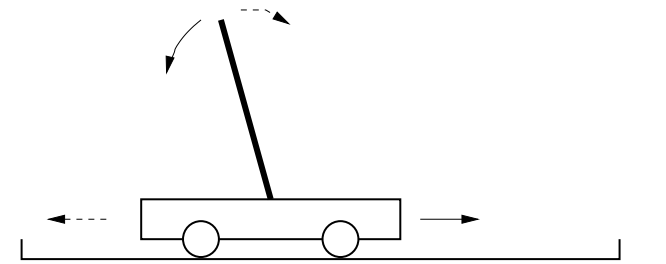
II. Motivation of using a Kohonen neural network

Kohonen networks are a class of neural networks known to solve the pole balancing problem [9]. Pole balancing is a control benchmark historically used in mechanical engineering. It involves a pole placed on a cart via a joint allowing movement along a single axis. The cart is able to move along a track with a fixed length as represented in figure 1(a). The

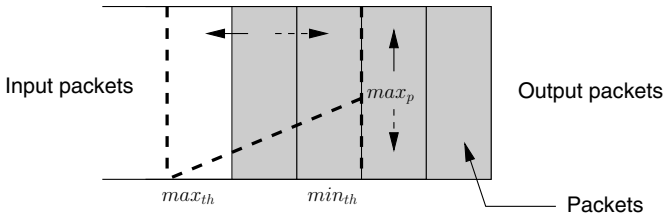
978-1-4244-2075-9/08/$25.00 ©2008 IEEE

aim of the problem is to keep this pole balanced by applying forces to the cart.

The main idea of our contribution is based on the analogy existing between this balancing problem and the RED queueing problem. In RED, we can compare the pole balancing to the evolution of the queue occupancy which oscillates between both thresholds (min_{th}, max_{th}). The physical forces resulting on the pole have a similar role to the packets arrival rate in the queue. Figure 1 illustrates this view.



(a) Pole balancing



(b) Adaptive RED

Fig. 1. Analogy between the single pole balancing problem and RED AQM

	Pole	RED
input_value[1]	previous position	previous queue length
input_value[2]	new position	current queue length
output_value[1]	force to apply in Newton	max_p

TABLE I
INPUT AND OUTPUT VALUES USED

Self-configuring RED schemes such as FRED or ARED update the max_p value as a function of the arrival rate in order to stabilize the queue size between both thresholds, min_{th} and max_{th}. In [2], the authors explain the queue length variation by the need of dynamically changing max_p as function of the queue occupancy. They propose to recompute this probability following an AIMD algorithm. The update is done as function of the average queue size. If the average queue size is around max_{th}, the algorithm increases max_p to drop more packets and decreases max_p if the value is around min_{th}.

The AIMD algorithm performed by FRED is different from ARED. Indeed, FRED updates max_p each time a packet is enqueued while ARED has another parameter allowing to update this value during a time interval. This action period can smooth the effect of an aggressive setting of the AIMD factors. Moreover, FRED does not apply consecutive decrease or consecutive increase of the max_p value. This choice can be problematic in case of rapid traffic change.

The neural network we use here is known as the Kohonen Self Organizing Map (SOM) [7]. It consists in a one or two dimensional information processing layer of functional entities called neurons. It is connected to input data seen as input vectors and provides output data also as vectors. We present in table I the entries used to feed the neural network in both cases and the resulting output. The input vector contains the previous and the current queue length and the output vector the max_p probability. For a sake of comparison, we give in this table the vectors used with the pole balancing problem. The input data is fully mapped onto the Kohonen layer's neurons which respond to this data according to the weight assigned to the connexions between input vectors and neurons and deliver an output response vector. To begin with, the neural network is presented a learning set of example input vectors and adjusts (i.e. learns) appropriate weights for its neurons by comparing the input vectors to the weight vectors for each neuron thus electing a "winning" neuron "close" to the input vector.

In addition to this, the Kohonen SOM deals with a topological learning feature, which implies neural neighborhood generalization of a correct learning experience so as to create clusters of neurons responding to similar input vectors without necessarily having explicitly learnt them. If a neuron learns that a given input vector is a vector it should respond to, its neighbours will learn they also should respond, only in a lesser way, depending on their topological distance to the first "winning" neuron. This way, the Kohonen SOM is well adapted to stability preservation tasks as the one we present here. Once the learning procedure is over, i.e. when the neural network produces an acceptable amount of erroneous responses during learning, the weights of the neural connexions to the data input are freezed. That means that the training process needs to be done only once without specific scenario and should work for every kind of situation.

Given the Kohonen SOM algorithm, the neural network can generalize its learnt experiences to other input vectors it has never seen before and produce adapted responses. In this way, the conservation of a direction, an equilibrium or the correct parameter to adjust a RED mechanism is made possible although there is no way of predicting the way the neural network learns to solve this particular problem. In our case, the learnt sequences of input vectors are not the ones used in our tests, in order to prove that the learning method provides a general purpose neural network for the resolution of the problem we deal with here. Once it has learnt, it can be used indefinitely for the task it has been trained for.

Previous related work [3] presents the use of a multi layer perceptron to adapt the α and β coefficients of a PI controller. In [3], the authors don't improve the queue length stability but smooth the PI dynamic and in average, results obtained

are globally similar. We think that such a neural network is well adapted to pattern and shape recognition problems, whilst a SOM such as the Kohonen SOM could be better suited to the task of stability preservation which we deal with here. Indeed, this Kohonen SOM algorithm preserves topological relationships between neighbouring vectors.

Each time a packet is enqueued, the Kohonen network computes a new max_p following the previous and the current average queue size. No other parameters are needed to perform this operation.

III. Implementation

One important point of dealing with Kohonen network is the small memory footprint required by the implementation. In our case, we have implemented our proposal in ns-2 simulator. The most complex structure is simply a square matrix 25×25 which represents the Kohonen network. The code used is a modification of the well-known Karsten Kutza's implementation[1]. All the scripts and ns-2 implementation used in this study are available for download at the author's webpage[2].

IV. The training process

The KRED queue has been trained with an arbitrary chosen number of eight long-lived TCP/Newreno flows emitted during 600 seconds without traffic variation on a single link topology. The neural network map learnt to stabilize the KRED queue with the common parameters given table II after 331 seconds. No further training has been done. The resulting Kohonen map is used thereafter in all the experiments. Both experiments related in section V use the same Kohonen map resulting from the same training process.

V. Evaluation and analysis

This section presents the experiments driven to evaluate the KRED and comments the results obtained.

A. Testbed and assumptions

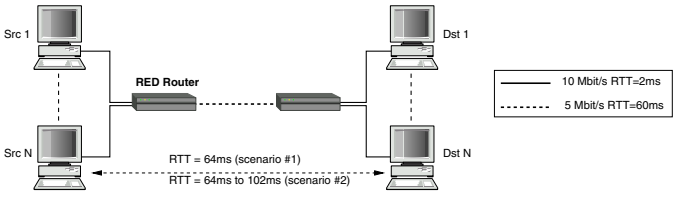
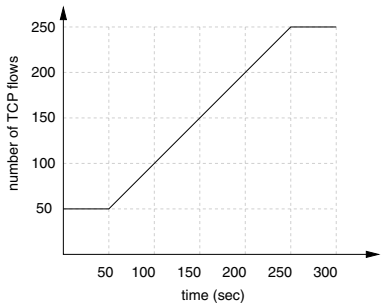


Fig. 2. The simulation topology

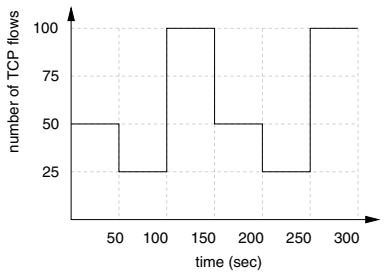
We drive experiments over a standard dumbbell topology represented in figure 2. We compare our proposal to RED, Feng RED (FRED), ARED and PI AQM. The parameters used for each queue are given in table II. The TCP flows are NewReno with a large window size set to 10000 packets. The RED queue is configured to drop and not to mark packets. In order to evaluate our proposal, we drive two distinct experiments. In the first scenario, the number of TCP flows in the network is increasing from 50 to 250 flows following the pattern figure 3(a). The RTT for each flow is identical. This scenario allows us to verify the impact of the traffic load on our proposal compared to others AQM. In the second experiment, the traffic changes every 50 seconds following the scenario presented in figure 3(b). Furthermore, each flow has a random RTT ranging from 64 to 102 ms. The rationale for using this traffic pattern is to evaluate our proposal under wide traffic variations.

[1]http://www.neural-networks-at-your-fingertips.com/

[2]http://manu.lochin.org/kred



(a) First scenario



(b) Second scenario

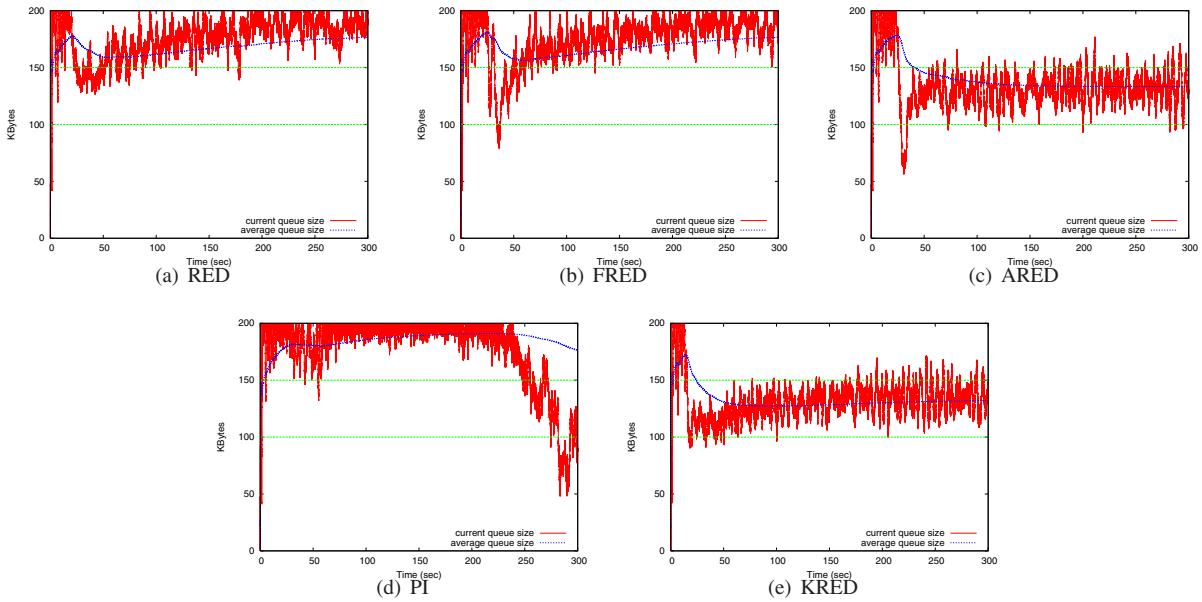
Fig. 3. The simulation scenarios

Common Parameters (C.P.)	$min_{th} = 100pkts$ $max_{th} = 150pkts$, $q_{size} = 200pkts$, $q_{weight} = 10^{-4}$.
RED	C.P., $max_p = 0.1$.
FRED	C.P., $max_p = 0.1$ $\alpha = 3.0$, $\beta = 2.0$.
ARED	C.P., $\alpha = 0.01$, $\beta = 0.09$, $gentle = true$, $interval = 0.3$, $max_p = 0.1$.
PI	$a = 1.822.10^{-5}$, $b = 1.816.10^{-5}$, $q_{ref} = 100pkts$, $w = 170Hz$.
KRED	C.P.

TABLE II
RED parameters used

B. First scenario

Results are given in figure 4. Each graph shows the instantaneous and average queue size. The two horizontal lines represent the min_{th} and max_{th} threshold RED parameters.



(a) RED (b) FRED (c) ARED (d) PI (e) KRED

Fig. 4. Performance comparison of various AQM with KRED, (1st scenario)

The results presented for the KRED queue are obtained after the training process. As shown in these figures, ARED 4(c) and KRED 4(e) queues obtain a stable queue length between both thresholds compared to the others. To better illustrate the benefit of our algorithm, we use as comparative metric the queue delay. The link statistics reported table III show that the queueing delay for ARED and KRED are the lowest. The drop rate for both queue is similar and the overall throughput at the output link is equal for all AQM.

With this first experiment we can conclude that ARED and KRED are the best algorithms in terms of stability of the queue when the traffic load increases but we cannot stand in favor of KRED since the results obtained are in the same order of magnitude. Indeed, the overall performances obtained by both AQM are quite similar. However, we have to keep in mind that fixed initial parameters are needed for ARED (given in table II).

The second scenario, presented in the next section, extends these measurements by changing the traffic pattern during the simulation and the RTT of each flow is ranging from 64 to 102 ms. The original parameters remain unchanged in order to verify the well adaptability of these AQM to the rapid traffic change problem.

C. Second scenario

As shown in figure 5, the KRED queue obtains a stable queue length between both thresholds compared to the AIMD process method of the FRED 5(b) and ARED 5(c). Moreover, KRED reacts rapidly to a traffic change compared to ARED. Due to the rapid traffic changing, the max_p value is constantly recomputed and the previous computed value strongly impacts on the current result. In the case of an AIMD process to compute the best max_p value, if the weights are small, the pace of convergence to the optimal value is slow and if the weights are high, the resulting probability can strongly oscillate when the traffic is changing. The initial configuration parameters used with success in the first scenario by ARED are not adapted to the second one. Thus, the KRED proposal allows to overcome this difficult problem of initial setting which is managed by the neural network. Finally, table IV give the statistics of this scenario and show that KRED still obtains the lowest average queuing delay.

AQM	Mean / Std. Dev. Queue Delay (ms)	Mean / Std. Dev. Link Throughput (Mbit/s)	TCP drop rate
RED	29.17 / 3.05	4.9978 / 0.29	22.23%
FRED	29.09 / 3.55	4.9978 / 0.29	22.39%
ARED	22.24 / 3.65	4.9978 / 0.29	25.23%
PI	29.51 / 5.27	4.9979 / 0.29	22.53%
KRED	21.92 / 3.02	4.9978 / 0.29	25.34%

TABLE III
STATISTICS FROM 1ST SCENARIO

AQM	Mean / Std. Dev. Queue Delay (ms)	Mean / Std. Dev. Link Throughput (Mbit/s)	TCP drop rate
RED	23.68 / 5.68	4.9978 / 0.0294	7.65%
FRED	21.96 / 6.14	4.9978 / 0.0294	8.54%
ARED	22.73 / 6.27	4.9978 / 0.0294	8.42%
PI	21.67 / 10.78	4.9979 / 0.0294	8.70%
KRED	20.10 / 4.75	4.9978 / 0.0294	9.15%

TABLE IV
STATISTICS FROM 2ND SCENARIO

VI. DISCUSSION AND CONCLUSIONS

This paper introduces Kohonen-RED: an adaptive RED mechanism easily implementable. The idea deals with the

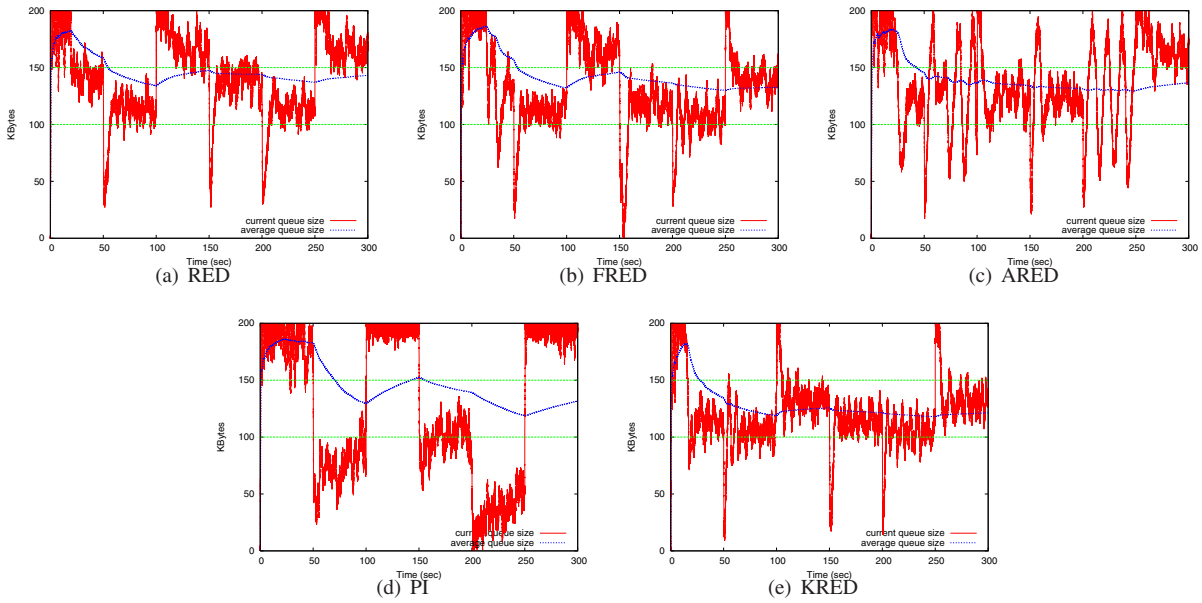


Fig. 5. Performance comparison of various AQM with KRED (2nd scenario)

use of a Kohonen neural network to compute the optimal probability parameter in order to achieve stable queue length. KRED reduces the number of parameters and in particular the non obvious ones. The Kohonen network does not need to be retrained and therefore can be put in hardware in the context of a router implementation. The mechanism's efficiency has been illustrated through ns-2 simulation where other schemes fail. In this work, we use a Kohonen based neural network specifically designed to solve the pole balancing problem. One of the main contribution of this study is to show the feasibility of using neural network to solve a networking stability problem. Considering promising preliminary results, we are currently designing a specific neural network for RED queue able to stabilize on a given value and not between two bounds and we are investigating on the improvements required in the core mechanism itself to achieve this goal. We also explore the design of a neural network able to accurately characterize the TCP behaviour.

ACKNOWLEDGMENTS

The authors would like to thank Sebastien Ardon and Guillaume Jourjon and Max Ott for the discussion about this mechanism and the support of the National ICT Australia (NICTA).

REFERENCES

[1] Robert Beverly and Karen Sollins. The role of learning in network architecture. Research Abstract of the Computer Science and Artificial Intelligence Laboratory (CSAIL) - http://publications.csail.mit.edu/abstracts/abstracts07/beverly2/beverly2.html.

[2] Wu chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shin. A self-configuring RED gateway. In *Proceedings of INFOCOM 99*, volume 3, pages 1320–1328, 1999.

[3] H.C. Cho, M.S. Fadali, and Hyunjeong Lee. Neural network control for tcp network congestion. In *Proc. of the 2005 American Control Conference*, pages 3480–3485, June 2005.

[4] S. Floyd, R. Gummadi, and S. Shenker. Adaptive red: An algorithm for increasing the robustness of red, technical report, international computer science institute, August 2001.

[5] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. 1(4):397–413, August 1993.

[6] C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. On designing improved controllers for AQM routers supporting TCP flows. In *INFOCOM*, pages 1726–1734, 2001.

[7] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Information Sciences*. Third extended edition edition, 2001.

[8] Steven H. Low, Fernando Paganini, Jiantao Wang, and John C. Doyle. Linear stability of tcp/red and a scalable control. *Computer Networks*, 43(5):633–647, 2003.

[9] A. Makarovic. Machine intelligence 12: towards an automated logic of human thought. In *Clarendon Press, New York, NY, USA*, pages 241–258, 1991.

[10] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99), London*, pages 260–262, 1999.

[11] T. Ziegler, S. Fdida, and C. Brandauer. Stability criteria of RED with TCP traffic. In *IFIP ATM&IP Working Conference*, Budapest, June 2001.